VisualBasic does I/O

Jon Titus, *Test & Measurement World*, Newton, MA

BBS

Microsoft's VisualBasic—which runs under Windows—cannot operate on a PC's I/O ports. However, you can add dynamic-link libraries (DLLs) to extend the keywords of native VisualBasic (**Listing 1**). The compiled code, listings, and documentation in ZIPfile DI1167Z.ZIP (attached to EDN BBS /DI_SIG #1367) let you perform I/O operations using the new keywords *Inp* and *Out*. These added keywords operate the same way these keywords do in other Basic dialects.

To use *Inp* and *Out* in VisualBasic, you must tell your program that these operations exist. Use the *Declare* statements, as **Listing 2** shows. Remember that the *Out* operation is a subroutine because it simply outputs a byte; it returns nothing back to your VisualBasic program. The *Inp* operation is a function, because it returns a byte from an input port. You use the *ByVal* keyword to tell VisualBasic that you want it to transfer the actual value, and not a reference or pointer to the value.

In addition, you must explicitly tell VisualBasic where to find the library functions in your DLL. This example assumes the DLL is on a floppy disk in the B drive. You can put the DLL on another drive, but be sure to tell VisualBasic where it is.

Simple VisualBasic program fragments show how to use the *Inp* and *Out* instructions. For example, Out 771, 130 sends the value 130 to output port 771 of my PC. On the other hand, A=Inp(769) assigns the byte at input port 769 to variable A. The syntax is exactly the same as in most other Basics.

EDN BBS /DI_SIG #1367 EDN

To Vote For This Design, Circle No. 343

Listing 1—C++ source code for DLL (Borland C++ compiler)

```
#include <d:\borlandc\include\windows.h>
#include <d:\borlandc\include\dos.h>

extern "C" {
int FAR PASCAL _export Test (int) ;
void FAR PASCAL _export Out (int, int) ;
int FAR PASCAL _export Inp (int) ;
}

int FAR PASCAL LibMain (HANDLE hInstance, WORD wDataSeg,
                WORD wHeapSize, LPSTR lpCmdLine)

     {
     hInstance = hInstance;
     wDataSeg = wDataSeg;
     wHeapSize = wHeapSize;
     lpCmdLine = lpCmdLine;

     if(wHeapSize > 0)
          UnlockData (0);
     return 1 ;
     }

int FAR PASCAL _export Test (int arg1)
{
     return (arg1 + 1);
}

void FAR PASCAL _export Out (int portaddr, int portdata)
{
     unsigned char data;
          outportb(portaddr,portdata);
}

int FAR PASCAL _export Inp (int portaddr)
{
     int portdata;
     portdata = inportb(portaddr);
     return (portdata);
}
```

Listing 2—VisualBasic location declarations

```
Declare Function Test Lib "b:\cuser2.dll" (ByVal Numb%) As Integer
Declare Sub Out Lib "b:\cuser2.dll" (ByVal Addr%, ByVal Byte%)
Declare Function Inp Lib "b:\cuser2.dll" (ByVal Addr%) As Integer
```

Motor-drive algorithm saves space and cycles

José A P Machado da Silva, University of Porto, Porto, Portugal

The algorithm embodied in the second subroutine in **Listing 1** generates the excitation sequence for most permanent-magnet and hybrid stepper motors. This subroutine is smaller than subroutines that spring from other algorithms.

To understand the algorithm, first consider that stepper motors have two stator coils, A and B, each having a center tap. Phase notation (A,A,B, and B) shows the direction of the current flow. That is, $AA=10_2$ symbolizes that current flows through half-coil A and that half-coil A is off.

To get the maximum torque from a stepper motor, you must drive two phases at a time. Using the binary notation developed in the preceding paragraph, the 4-phase drive sequence for all four half coils is 0101_2, 0110_2, 1010_2, 1001_2 or 5_{HEX}, 6_{HEX}, A_{HEX}, 9_{HEX}.

Listing 1—Stepper-motor drive subroutines

```
Memory Accessing
                                        bytes/cycles
Begin   MOV  Rr,curraddress               2/2       Load adr reg w/current
                                                      table address
New     MOV  Outpur,@ Rr                  2/2       Send drive sig to output
        INC  Rr                           1/1       Increment adr reg
        CJNE Rr,#lastaddress+1,More       3/2       Rel-cond jump to More
More    ...                               ...       Delay
        JNZ  New                          2/2       End of motion?
End     MOV  curraddress,Rr               2/2       Save adr reg

Accumulator Rotating

Begin   MOV  A,currstate                  2/1       Load A w/current drive
                                                      bytes
Next    MOV  Output,A                     2/1       Send drive sig to output
        RL   A                            1/1       Rotate A left twice
        RL   A                            1/1
        ...                               ...       Delay
        JNZ  Next                         2/2       End of motion?
End     MOV  curstate,A                   2/1       Save drive bytes
```